Introduction to Data Management
CSE 344

Lecture 24: MapReduce
HW8 is out

• Last assignment!
  – Get Amazon credits now (see instructions)

• Spark with Hadoop

• Due next wed
Parallel Data Processing @ 1990
Review: Shared Nothing

• Cluster of machines on high-speed network
• Called "clusters" or "blade servers"
• Each machine has its own memory and disk: lowest contention.

NOTE: Because all machines today have many cores and many disks, then shared-nothing systems typically run many "nodes" on a single physical machine.

Characteristics:
• Today, this is the most scalable architecture.
• Most difficult to administer and tune.

We discuss only Shared Nothing in class
Review: Approaches to Parallel Query Evaluation

- **Inter-query parallelism**
  - Transaction per node
  - OLTP

- **Inter-operator parallelism**
  - Operator per node
  - Both OLTP and Decision Support

- **Intra-operator parallelism**
  - Operator on multiple nodes
  - Decision Support

We study only intra-operator parallelism: most scalable
Distributed Query Processing

• Data is horizontally partitioned on many servers

• Operators may require data reshuffling
Horizontal Data Partitioning

Data:

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Servers:

1

2

...
Horizontal Data Partitioning

Data:

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Servers:

1. 

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. 

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... 

P

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which tuples go to what server?
Horizontal Data Partitioning

- **Block Partition:**
  - Partition tuples arbitrarily s.t. \( \text{size}(R_1) \approx \ldots \approx \text{size}(R_p) \)

- **Hash partitioned on attribute A:**
  - Tuple \( t \) goes to chunk \( i \), where \( i = h(t.A) \mod P + 1 \)

- **Range partitioned on attribute A:**
  - Partition the range of \( A \) into \( -\infty = v_0 < v_1 < \ldots < v_p = \infty \)
  - Tuple \( t \) goes to chunk \( i \), if \( v_{i-1} < t.A < v_i \)
Parallel Group By

Data: $R(K,A,B,C)$
Query: $\gamma_{A,\text{sum}(C)}(R)$

How to compute if:

- $R$ is hash-partitioned on $A$
- $R$ is block-partitioned
- $R$ is hash-partitioned on $K$
Parallel Group By

Data: $R(K,A,B,C)$

Query: $\gamma_{A,\text{sum}(C)}(R)$

• $R$ is block-partitioned or hash-partitioned on $K$

Reshuffle $R$ on attribute $A$
Parallel Join

- **Data:** \( R(K_1, A, B), S(K_2, B, C) \)
- **Query:** \( R(K_1, A, B) \bowtie S(K_2, B, C) \)

Initially, both \( R \) and \( S \) are horizontally partitioned on \( K_1 \) and \( K_2 \)

- Reshuffle \( R \) on \( R.B \) and \( S \) on \( S.B \)
- Each server computes the join locally
Data: $R(K_1, A, B), S(K_2, B, C)$
Query: $R(K_1, A, B) \bowtie S(K_2, B, C)$
Speedup and Scaleup

• Consider:
  – Query: $\gamma_{A,\text{sum}(C)}(R)$
  – Runtime: dominated by reading chunks from disk

• If we double the number of nodes $P$, what is the new running time?
  – Half (each server holds $\frac{1}{2}$ as many chunks)

• If we double both $P$ and the size of $R$, what is the new running time?
  – Same (each server holds the same # of chunks)
Uniform Data v.s. Skewed Data

• Let \( R(K,A,B,C) \); which of the following partition methods may result in skewed partitions?

• Block partition
  
• Hash-partition
  – On the key \( K \)
  – On the attribute \( A \)

Uniform

May be skewed

Assuming good hash function

E.g. when all records have the same value of the attribute \( A \), then all records end up in the same partition
Loading Data into a Parallel DBMS

Example using Teradata

AMP = “Access Module Processor” = unit of parallelism
Example Parallel Query Execution

Find all orders from today, along with the items ordered

```
SELECT *
FROM Order o, Line i
WHERE o.item = i.item
AND o.date = today()
```
Example Parallel Query Execution

Order(oid, item, date), Line(item, …)
Example Parallel Query Execution

\[\text{Order}(\text{oid, item, date}), \text{Line}(\text{item, …})\]
Example Parallel Query Execution

Order(oid, item, date), Line(item, …)

AMP 1

AMP 2

AMP 3

join: o.item = i.item

contains all orders and all lines where hash(item) = 1

contains all orders and all lines where hash(item) = 2

contains all orders and all lines where hash(item) = 3
Parallel Data Processing @ 2000
Optional Reading

• Original paper: https://www.usenix.org/legacy/events/osdi04/tech/dean.html

• Rebuttal to a comparison with parallel DBs: http://dl.acm.org/citation.cfm?doid=1629175.1629198

• Chapter 2 (Sections 1,2,3 only) of Mining of Massive Datasets, by Rajaraman and Ullman http://i.stanford.edu/~ullman/mmds.html
Distributed File System (DFS)

- For very large files: TBs, PBs
- Each file is partitioned into chunks, typically 64MB
- Each chunk is replicated several times (≥3), on different racks, for fault tolerance
- Implementations:
  - Google’s DFS: GFS, proprietary
  - Hadoop’s DFS: HDFS, open source
MapReduce

- Google: paper published 2004
- Free variant: Hadoop
- MapReduce = high-level programming model and implementation for large-scale parallel data processing
Typical Problems Solved by MR

- Read a lot of data
- **Map**: extract something you care about from each record
- Shuffle and Sort
- **Reduce**: aggregate, summarize, filter, transform
- Write the results

Paradigm stays the same, change map and reduce functions for different problems
Map Reduce Data Model

**Instance**: Files!
- where a file = a bag of \((key, value)\) pairs

**Schema**: None!
- just like other key-value data models

**Query language**: a MapReduce program:
- Input: a bag of \((inputkey, value)\) pairs
- Output: a bag of \((outputkey, value)\) pairs
Step 1: the MAP Phase

User provides the MAP-function:

- Input: \((\text{input key, value})\)
- Output: bag of \((\text{intermediate key, value})\)

System applies the map function in parallel to all \((\text{input key, value})\) pairs in the input file
Step 2: the **REDUCE** Phase

User provides the **REDUCE** function:

- **Input:** (intermediate key, bag of values)
- **Output:** bag of output (values)

System groups all pairs with the same intermediate key, and passes the bag of values to the **REDUCE** function
Example

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The **key** = document id (did)
  - The **value** = set of words (word)

```java
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += parseInt(v);
  Emit(AsString(result));
```
MAP

- (did1, v1)
- (did2, v2)
- (did3, v3)
  ...

REDUCE

- (w1, 1)
- (w2, 1)
- (w3, 1)
  ...
- (w1, 25)
- (w2, 77)
- (w3, 12)
  ...

Shuffle

CSE 344 - Fall 2016
Jobs v.s. Tasks

• A MapReduce Job
  - One single “query,” e.g., count the words in all docs
  - More complex queries may consist of multiple jobs

• A Map Task, or a Reduce Task
  - A group of instantiations of the map-, or reduce-function, which are scheduled on a single worker